DOCUMENT RESUME

ED 459 816 IR 058 352

AUTHOR Laleuf, Jean R.; Spalter, Anne Morgan
TITLE A Component Repository for Learning Objects: A Progress Report.
SPONS AGENCY National Science Foundation, Arlington, VA.
PUB DATE 2001-00-00
NOTE 10p.; In: Proceedings of the ACM/IEEE-CS Joint Conference on Digital Libraries (1st, Roanoke, Virginia, June 24-28, 2001). For entire proceedings, see IR 058 348. Sponsored by NSDL and STC grants, and Sun Microsystems. Figures may not reproduce well.
AVAILABLE FROM Association for Computing Machinery, 1515 Broadway, New York NY 10036. Tel: 800-342-6626 (Toll Free); Tel: 212-626-0500; e-mail: acmhelp@acm.org. For full text: http://www1.acm.org/pubs/contents/proceedings/dl/379437/.
PUB TYPE Reports - Research (143) -- Speeches/Meeting Papers (150)
EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Computer Graphics; Computer Software Development; Educational Environment; Educational Technology; *Electronic Libraries; Instructional Materials; Introductory Courses; Library Materials; *Material Development; Programming

ABSTRACT

An important category of SMET digital library content will be highly interactive, explorable microworlds for teaching science, mathematics, and engineering concepts. Such environments have proved to be extraordinarily time-consuming and difficult to produce, while threatening the goals of widespread creation and use. One proposed solution for accelerating production has been the creation of repositories of reusable software components or learning objects. Programmers would use such components to rapidly assemble larger-scale environments. Although many agree on the value of this approach, few repositories of such components have been successfully created. This paper suggests some reasons for the lack of expected results and proposes two strategies for developing such repositories. A case study is reported that provides a proof of concept of these strategies. (Contains 20 references.) (Author/AEF)

Reproductions supplied by EDRS are the best that can be made
from the original document.

ED 459 816

A Component Repository for Learning Objects: a Progress Report

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL HAS BEEN GRANTED BY

D. Cotton

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

1

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

By: Jean R. Laleuf & Anne Morgan Spalter

IR058352

BEST COPY AVAILABLE

A Component Repository for Learning Objects

A Progress Report

Jean R. Laleuf
Brown University
Department of Computer Science Box 1910
(401) 863-7658
jrl@cs.brown.edu

Anne Morgan Spalter
Brown University
Department of Computer Science Box 1910
(401) 863-7615
ams@cs.brown.edu

ABSTRACT

We believe that an important category of SMET digital library content will be highly interactive, explorable microworlds for teaching science, mathematics, and engineering concepts. Such environments have proved extraordinarily time-consuming and difficult to produce, however, threatening the goals of widespread creation and use.

One proposed solution for accelerating production has been the creation of repositories of reusable software components or learning objects. Programmers would use such components to rapidly assemble larger-scale environments. Although many agree on the value of this approach, few repositories of such components have been successfully created. We suggest some reasons for the lack of expected results and propose two strategies for developing such repositories. We report on a case study that provides a proof of concept of these strategies.

Keywords

Components, design, digital library, education, learning objects, NSDL, reuse, software engineering, standards.

1. INTRODUCTION

Our vision for digital library content goes beyond scanned literature or searchable curriculum materials to include richly interactive explorable microworlds that take full advantage of the ever-increasing power of computers, software, and networks. These learning environments combine the best qualities of live demonstration (see Fig. 1) with interaction only possible on the computer. Unfortunately, our research has led us to the conclusion that it is an unexpectedly huge effort to create a complete collection of interactive learning experiences for even a single introductory course in a given discipline.

We have been trying to accelerate the development process by creating reusable components or learning objects that can be recombined in different ways to produce sets of learning environments. By components or learning objects we mean standardized pieces of code, usually class files or Java beans, which programmers can easily reuse in different programs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
JCDL '01, June 24-28, 2001, Roanoke, Virginia, USA.
Copyright 2001 ACM 1-58113-345-6/01/0006...$5.00.

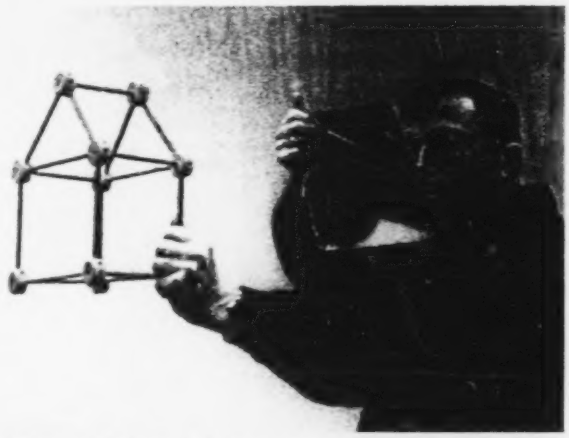

Figure 1: Professor van Dam uses a Tinkertoy house and a cardboard perspective viewing volume to demonstrate camera viewing transformations.

It may seem at first that creating a few dozen components would suffice for many courses. For example, an introductory calculus course would have a function-graphing component, some tools for finding derivatives and integrals, a series expander, expression editors and a few more objects, but the reality is far more complicated. The situation is similar to that faced by GUI designers. If one looks at an interface, it seems to be made up of a few basic elements, such as field boxes, sliders, buttons, and panes, but GUI libraries are huge and take enormous effort to develop.

Based on the GUI library analogy, we believe that a half a dozen programmers and researchers could never construct a major reusable educational components library in a reasonable amount of time. Only a concerted collaborative effort of tens of person-years can build the necessary content for any given field. In particular, although some underlying components, such as math libraries for learning objects, may be applicable across many science education domains, more domain-specific components must be created separately for each field.

The GUI library analogy assumes that we know exactly how to make these components, but in the field of educational software components there are many open research issues. For example, how does one analyze current simulations for decomposition into reusable components? How can one design components to be

useful for educators (as well as programmers)? And how does one choose a proper level of granularity for a component?

We have been exploring two main strategies for repository creation. The first is the use of a categorization scheme to help programmers analyze and characterize types of components as they relate to educational purposes. The second is a method for addressing issues of object granularity, and determining what levels of object complexity are appropriate. We have applied these strategies in a proof-of-concept case study.

The work described here is part of an NSF NSDL grant, the CREATE project (A Component Repository and Environment for Assembly of Teaching Environments).

2. PREVIOUS WORK

It has long been hoped that instructional software developers would contribute to libraries of instructional software that would give others access to the benefits of software simulations and interactive exercises. There are a number of significant efforts to create develop and share educational software components, most of them based on the Java language. The ESCOT project is a testbed that seeks to encourage development and reuse of learning objects, with a current emphasis on middle-school mathematics, but their components are not available for general public use [6]. The E-Slate company sells educational components (as opposed to finished applications) and describes about two dozen of them on their Web site [7]. The Educational Object Economy project compiles interactive educational tools in the form of complete applets [5]. As far as we can tell, none of these undertakings provides complete sets of components for specific courses and even if they become quite successful, are aimed chiefly at educators with little or no programming experience. There is still a need for the digital library to house lower-level components, to be used by programmers to create fully customized educational environments. Further discussion of educational component use can be found in the IEEE Computer September 1999 Special Issue on Web based learning and collaboration [13].

In addition to work specific to educational software, the technical, social, economic, and administrative problems with general computer code reuse are now better understood [4, 11]. Problems include failure to organize and index reusable objects, failure to mandate that code be designed with reuse in mind, lack of an organizational structure dedicated to supporting reuse, and failure to recognize the domain dependency of reuse strategies.

The Exploratories project at Brown University, on which this paper's work is based, has worked for over five years to create learning objects with high levels of interactivity [8]. Our chief content area has been introductory computer graphics, including introductory linear algebra [2, 17]. We think of exploratories as combinations of "exploratoriums" [9] and laboratories, realized as two- and three-dimensional explorable worlds which are currently implemented as Java applets. Our applets are embedded in a hypertext environment and are used by a number of high school- and college-level courses around the world. They are freely available at our Web site [8].

When we began trying to reuse frequently occurring program elements, we quickly found that simply copying and pasting code was not a good strategy. Most classes worked well only in the program for which they were initially designed. The problems for reuse ranged from a lack of software interface standards (such as those imposed by the Java beans spec [16]) to difficulty in arriving at the right level of feature complexity. In the end, we found that everyone wound up rewriting the "reusable" elements. The strategies and project discussed in this paper were inspired by this situation.

The Exploratories project has tried to promote other aspects of reusability by creating reusable hypertext structures for Web-based curricula [19], describing methods for integrating learning objects into traditional curricula [20], categorizing pedagogical approaches and teaching techniques that can be used for interactive learning environments [18], and creating a Web-based repository structure for JavaBeans [3].

2.1 A Component Categorization Strategy

When we began making components, we found that they fell naturally into three different categories. We characterize all reusable educational components as either 1. Core Technologies, which have a high degree of domain independence and are typically quite fine-grained (e.g., Java GUI classes), 2. Support Technologies, which usually have some domain dependence and are typically medium-grained objects (e.g., BEA Systems JavaBeans designed for e-commerce (includes shopping cart beans, order tracking beans, etc.), or 3. Application Technologies, which are almost always highly domain dependent and coarse-grained (e.g., a Java applet that teaches about a particular chemical reaction). These categories are described in more detail below:

2.1.1 Core Technologies

Characteristics

- Domain independence
- High levels of reusability
- Self-contained functionality
- Adherence to high standards of design and reliability

Audience

Chiefly programmers but also some content developers (with little or no programming ability) using assembly tools

Granularity

Typically fine-grained (e.g., sliders, timers and buttons) but can also include coarse-grained objects (e.g., spreadsheets or even an entire pedagogical framework into which one can plug one's materials [19])

Examples

- Java GUI classes
- Various math function libraries
- IBM AlphaBeans [1]
- 3D Interaction and visualization widgets

Notes

Many commercial offerings fall into this category, but such components may also be built in-house.

2.1.2 Support Technologies

Characteristics

- Domain dependence
- Moderate levels of reusability
- Self-contained functionality

Audience

Chiefly programmers but also some content developers (with little or no programming ability) using assembly tools.

Granularity

Typically medium-grained objects, often aggregating finer-grained components (e.g., an image filtering widget that combines slider and windowing components)

Examples

BEA Systems JavaBeans designed for e-commerce (e.g., shopping cart beans, order tracking beans, etc.)

2.1.3 Application Technologies

Characteristics:

- Domain dependence
- Minimal levels of reusability
- Each object is a self-contained, fully-functional application or applet that attempts to achieve an educational goal.

Audience:

End users without programming experience. Application Technology components can be combined by both programmers or non-programmers (using a visual environment) and can be part of a larger pedagogical structure (e.g., a game or lab) or a full-blown curriculum.

Granularity:

Typically coarse-grained, although useful distinctions can be made about the level of concept granularity of these end products (e.g., fine-grained applets teaching a single concept vs. coarse-grained applets teaching a number of concepts in one module)

Examples:

- An applet that teaches characteristics of a particular chemical reaction.
- An applet that teaches how to make a scenegraph in 3D graphics.

Notes:

These objects may also be part of a larger group of similar objects (e.g., a series of applets to teach increasingly complex topics in a subject). Because these objects attempt to achieve an educational goal and are targeted primarily at end-users, pedagogical, user interface, and information structure concerns are highly important in the creation of objects for this category.

Note that in comparison to other fields, especially e-commerce, educational technology is particularly deficient in the support technologies category. We know of no substantial repositories for such domain-specific components and therefore observe educational software developers repeatedly building these types of components. We believe that one of the chief problems facing current efforts in educational component technology is a lack of distinction between support and core technologies. This is compounded further by a deficiency in understanding the design features necessary to promote reuse of support technologies.

2.2 A Granularity Strategy

Although the categories described above provide guidance as to what levels of granularity are appropriate for specific types of components, individual components may in fact be of any granularity regardless of the category in which they fall. In our experience of creating components, we found that, in fact, creating them solely at any single level of granularity either meant having too many features and not enough flexibility, or having so fine-grained a set of components that a great deal of work still needed to be done to create the final product.

Our strategy for dealing with granularity and organizing the results is, for a given component, to produce a **complete** set of sub-components, thus providing objects at **all** levels of granularity, from application technology components (e.g., self-contained interactive applets) down to core technology code (e.g., a coordinate system package which includes a complete set of interfaces and behaviors for coordinate systems). This decomposition is important to complete even if there is no current need for some of the components it generates.

When this strategy is employed, programmers given a component from any category will be able to customize that component and also to retrieve, customize, reuse and reconfigure any sub-components it might aggregate. Teachers working with programmers would have virtually no constraints on how they could reconfigure what they see on the screen to meet their own needs. By including all levels of component granularity in the development process, we hope to create component repositories that will be broadly applicable to diverse areas of science education. In addition, completely decomposed components describe a hierarchy that is helpful for documentation purposes. The downside of this strategy is that it requires a large upfront investment, as discussed in the Conclusion.

3. Case Study: The Camera Viewing Transformation

Our case study is based on a set of applets that teach students in an introductory graphics course about 3D camera transformations.

One of the basic tasks in computer graphics is generating a representation of a three-dimensional scene and displaying it on the two-dimensional computer screen. This is performed in a manner very similar to positioning a camera in front of a real-world scene, taking a photograph, and looking at the resulting image.

For the sake of mathematical simplicity and efficiency, a series of manipulations must be performed on the scene and its geometry before its two-dimensional representation can be drawn on the screen. Pedagogically, these are best described as changes in

position and shape of the objects concerned. The continuum of these changes is of particular interest to students.

3.1 History

3.1.1 Text Illustrations

Despite the inherently continuous nature of the camera transformations material, the canonical reference text in computer graphics, Computer Graphics: Principles and Practice [10], provides only five pairs of discrete snapshots of these continuous manipulations (see Fig. 2). Unfortunately, these images are at best hard to decipher and are typically found confusing and unintuitive. The essential difficulty of these images is that they are preset, providing no opportunity for exploration or discovery through manipulation of the scene and the operations it undergoes.

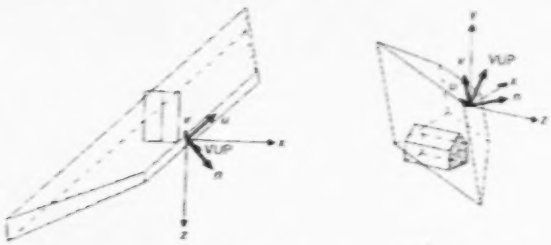

Figure 2: One of five pairs of diagrams used to illustrate the camera viewing transformations.

3.1.2 Models used in class

In his class on introductory computer graphics, Professor Andries van Dam does a lot to impart the continuous nature of the scene transformations through the use of Tinkertoy props, as shown in Fig. 1. This resolves many of the issues presented by the book's illustrations, as he is able to move the props around and explain how they move and change orientation in time. Students see the continuity of motion and more fully grasp the concepts presented to them. It is difficult, however, to show more than one model changing at a time and Tinkertoys lack the flexibility necessary to accurately display the concluding mathematical transformations. These include scaling and other non-rigid body, distorting transformations, requiring much hand-waving by the professor and implicit visualization by the students.

3.1.3 Customized software

One of the first attempts at using computer technology to resolve the problems above was a program written at Brown for an Evans and Sutherland vector display in the late '70s [12]. The resulting program allowed one to manipulate the parameters of the computer camera and then have the program animate the succession of transformations to produce a 2D image. Students observed the continuity of the changes and saw how one shape smoothly turned into another over time. In the early '90s the program was reimplemented for a raster display, again using an experimental software system.

This second version was more powerful and compelling than the first. It was heavily used for many years but had several limitations, such as the fact that it only ran in our research lab. In addition, because of the nature of the program design, it was impossible to easily modify it and when new viewing models were introduced in the class, the demo became somewhat confusing. As with all software, it also suffered from code rot. Again, the custom nature of the software made this difficult to address. When the underlying software system was abandoned by the research group, the program eventually ceased to function. A video was made, but the lack of interaction greatly diminished its usefulness.

3.2 Current incarnation

The idea of an interactive, computer-based educational tool describing the camera transformations was revisited in early fall of 1999 when one of the course's teaching assistants offered to throw together a rough prototype replicating a subset of the FLESH program's capabilities.

Working off this successful prototype, we engaged in a more formal design process, aiming to better think out the various pedagogical considerations and also use the new applet as a proving ground for our ideas on component design and organization. We thought through the potential components in terms of our three categories and began to decompose each one according to our theory of granularity completeness.

3.2.1 Pedagogical considerations

We sought primarily to provide a means by which a user could visualize the various manipulations undergone by a synthetic scene when its two-dimensional representation is generated. It needed to be useful for the professor demonstrating the concepts to his class during a lecture as well as accessible to students who wanted to revisit the material and explore and experiment on their own time.

To these ends we sought to have our new versions encompass all the capabilities of other methods of teaching the material (such as the diagrams and models) while enabling a students to freely explore and answer any question that might occur to them. We chose to provide an interactive, three-dimensional computer rendering to allow one to view the scene from multiple angles. We also gave the users complete control over the passage of time in the simulation, allowing them to advance or backtrack as needed to better understand a particular concept. Furthermore, we provided full, interactive and graphical control over the various camera parameters (position, orientation, field of view, etc.) at all stages of the simulation in order to offer students as many possibilities for exploration as possible.

3.2.2 Component structure

In Fig. 3, we see a sample of the components used in the current incarnation of the camera viewing applet. The transparent gray truncated pyramid represents the computer's perspective view volume, and the scene being viewed consists entirely of the simple house. A representative component structure, the *Vector Solid View*, has been selected to show how a coarse-grained, high-level component can be broken down into fine-grained, customizable components.

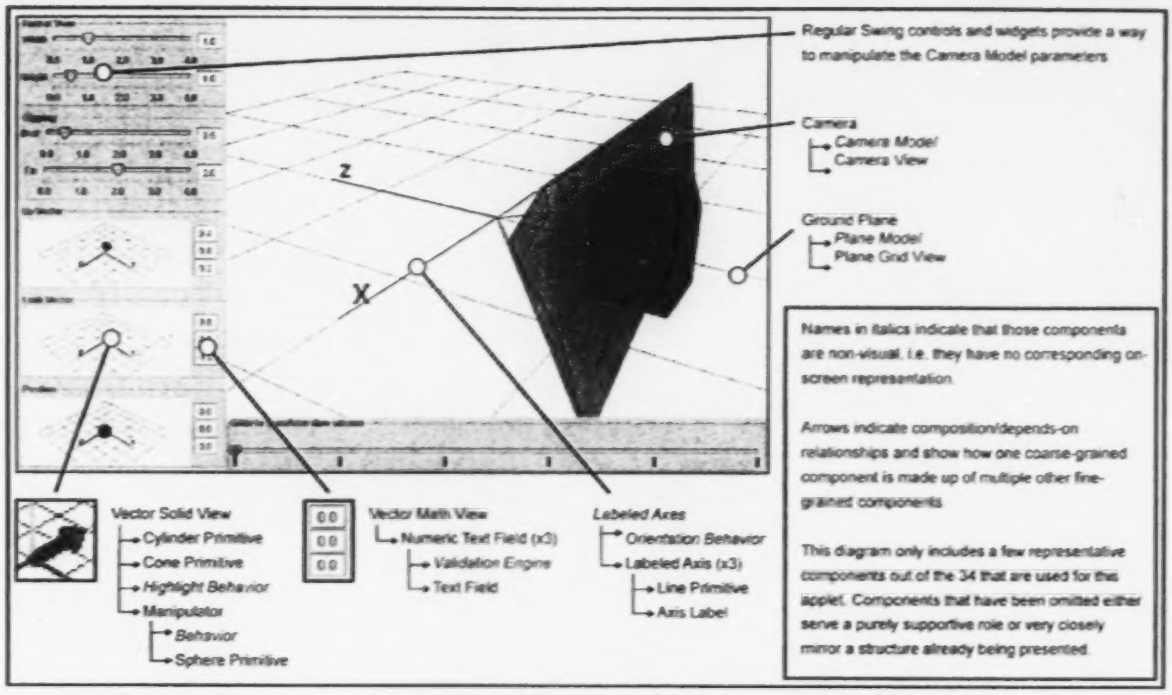
Regular Swing controls and widgets provide a way to manipulate the Camera Model parameters
Camera
Camera Model
Camera View
Ground Plane
Plane Model
Plane Grid View
Z
X
Names in italics indicate that those components are non-visual, i.e. they have no corresponding on-screen representation.
Arrows indicate composition/depends-on relationships and show how one coarse-grained component is made up of multiple other fine-grained components.
This diagram only includes a few representative components out of the 34 that are used for this applet. Components that have been omitted either serve a purely supportive role or very closely mirror a structure already being presented.
Vector Solid View
Cylinder Primitive
Cone Primitive
Highlight Behavior
Manipulator
Behavior
Sphere Primitive
Vector Math View
Numeric Text Field (x3)
Validation Engine
Text Field
Labeled Axes
Orientation Behavior
Labeled Axis (x3)
Line Primitive
Axis Label

Figure 3: Decomposition of a sample set of the components used in the camera viewing transformation applet.

Vector Solid View:

Cylinder primitive
The cylinder primitive provides a visual representation of the stem of the vector's arrow representation. It would be relatively simple to replace it with another primitive such as a rectangular solid.

Cone primitive
Serving a function similar to the cylinder primitive component, the cone primitive provides a visual representation of the vector arrow's head. As with the cylinder, changing the type of primitive used would be trivial.

Highlight Behavior
This non-visual component provides user feedback functionality by highlighting the head and stem of the vector arrow when the user's mouse passes over them. This allows users to intuitively understand that something will happen if they click on the different parts of the vector.

Manipulator
This component allows users to visually interact with the vector, dragging it around to change its orientation. It is composed of three different components, one of which has been omitted from the diagram for simplicity's sake but which will be described below.

Behavior
This component plugs into the Java3D interaction framework to allow users to interact with the vector by dragging it to the desired position.

Sphere Primitive
This object provides visual feedback to users while they drag the vector arrow to a new orientation.

Spherical Geometry Intersection Utility Class
Although omitted from the diagram, this non-visual component serves a critical role by performing the calculations necessary for the Behavior component to function correctly. Encapsulating this functionality in a single component facilitates long-term maintenance, allowing it to be replaced at a later date with a more efficient implementation should one be developed.

3.3 Reusability

Having decomposed the Camera Viewing Transformation Applet into a number of fine-grained components, one can now turn to the issue of these components' reuse. More specifically, one must look at whether or not these components are at all reusable outside of the one or two applets that were in the designer's mind when they were created.

Fig.s 4-6 show the component usage for three different classes of components. In Fig. 4, the camera viewing applets all reference the same set of components. Indeed, our four different viewing applets (Parallel Camera Parameters, Parallel Camera Transformation, Perspective Camera Parameters, and Perspective Camera Transformation [8]) each teach different aspects of the transformation process but use the same set of components, passing in different parameters to get the desired effects. Notice that the camera applets use a mixture of both core technology and support technology components. Also, some support technology components also reuse core technology components. For example, the Camera Interpolator component reuses all the components in the 3D Interpolators core component package.

In Fig. 5, we see that an applet that reuses many of the components used in the camera viewing applets. This applet helps students understand the rendering process called radiosity, in

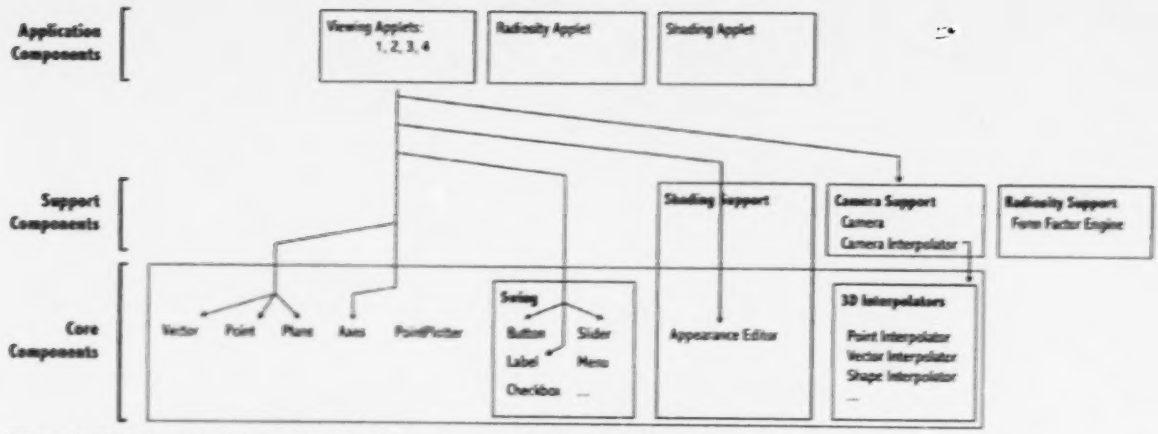

Figure 4: The component reuse graph for the four camera viewing transformation applets.

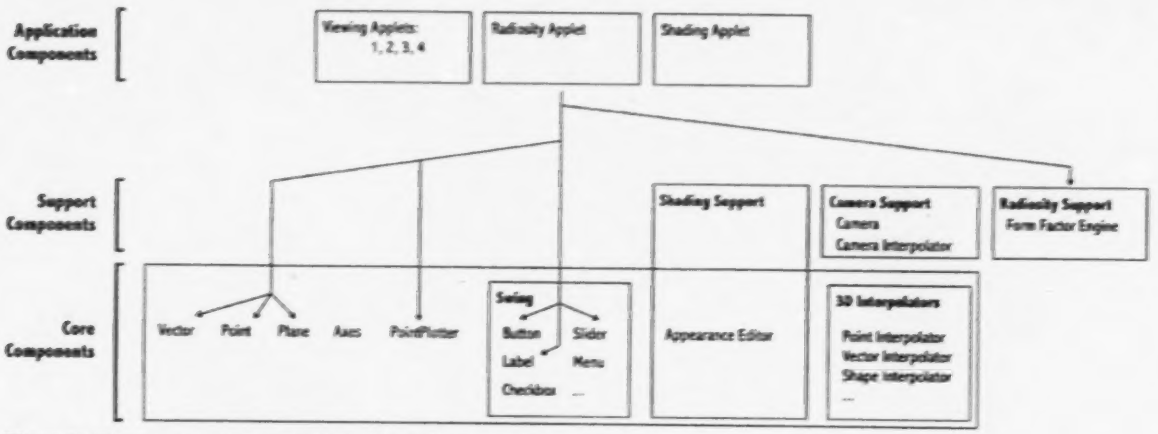

Figure 5: The component reuse graph for the radiosity applet.

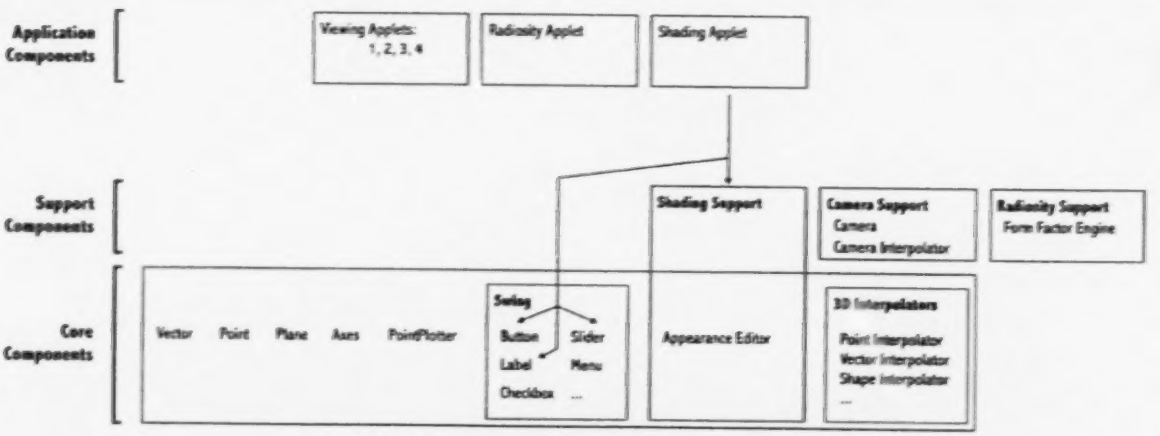

Figure 6: The component reuse graph for the shading applet.

BEST COPY AVAILABLE

which energy transport is simulated to calculate diffuse light reflections in a scene. We see reuse of the Vector, Point and Plane core technology components as well as the standard Swing components. The Axes component has not been reused because the PointPlotter component is being used instead. Notice also that although the components in the Camera Support package are not being used, those in the Radiosity Support package are. This is in keeping with the fact that support technology components are typically domain-specific and, although reusable for similar applets, are generally not reusable by applets from different domains.

Fig. 6 demonstrates that components can, in some cases, be classified into two different categories depending on how they are used by the application component. In this case, the Appearance Editor component is being used as a support technology component because it plays a central supportive role for the shading applet. Contrast this to Fig. 4, in which this very same component acts as a core technology component with respect to the camera viewing applets. This ability of components to migrate from role to role depending on how they are used by other components adds more flexibility to the classification schema laid out above and thereby enhances its power.

3.3.1 Standards

All of our components adhere to the JavaBeans specification because doing so facilitates integration of components into commercial software design packages. A properly-designed JavaBean, for instance, can be used without any problems in Sun's BeanBox, Inprise's JBuilder, Forte's Netbeans IDE, or IBM's VisualAge. It can be used with equal success in the educational authoring environments being produced by our colleagues at ESCOT and e-Slate. The universal acceptance of the JavaBeans standard therefore makes it a valuable requirement for all our component designs and will make our components that much more powerful down the road.

3.4 Future Work

Our digital library grant is a collaborative one with chemistry professor Dave Yaron at Carnegie Mellon. Professor Yaron and other members of the grant team at Carnegie Mellon have been creating environments in which non-programming educators can customize interactive experiences and embed them in their own pedagogical materials. We have begun to join our different approaches to the problem of reuse and customization in the digital library efforts in a collaborative project to aid chemistry education.

3.4.1 Molecular Visualization Applet

We are working on a molecular visualization applet to increase the reuse of existing components as well as introduce new components developed in-house and by third parties.

A current prototype aims to reuse the plane model and view components as well as some of the core 3D object manipulation components that allow users to intuitively move objects in a 3D space.

This prototype expands the core technology space by using a VRML file loader imported from a third party source and using it to import VRML models retrieved from the Protein Data Bank, thereby leveraging content from an existing digital library. We are also planning on introducing a precise collision detection package to the core packages as well as more basic 3D interaction widgets.

Finally, we will provide a protein docking engine component that might find significant reuse in other software dealing with protein interactions.

The resulting set of components will be glued together using our Carnegie Mellon collaborators' non-programming environment, thereby providing the power of various in-house and third-party software components in an easy to use yet powerful building tool [15].

3.4.2 Increased Compatibility Effort

Recognizing that it would be a grave mistake to isolate our work from complementary research and development performed by other research groups, we will also be maximizing compatibility between the components we develop and development environments and test beds such as those produced by the E-Slate and ESCOT projects. Recent developments have made these environments more compatible with the JavaBeans standard and we anticipate increased productivity by leveraging the tools they provide.

3.4.3 Metadata Standards for Harvesting

We have started tagging the applets we produce with standard IMS [14] metadata tags in order to make our efforts easily harvestable by collectors of digital library material. In the near future, we intend to extend this effort by tagging individual components, thereby allowing harvesters into our component libraries as well as our applet catalogs.

4. DISCUSSION AND CONCLUSION

When we proposed our framework and granularity strategy as part of our grant application, we had limited experience using either one for real-life applications. By creating the camera viewing transformation applet set, we were able to test out these theories in practice. We feel that the results offer a proof of concept of these strategies.

We have found, however, that there are also disadvantages to using a structured, component-centric approach with an emphasis on reusability. Although the up-front design results in components that are well designed, this design methodology greatly increases the length of the development cycle. For example, the viewing techniques applets used in our case study took roughly four months to design, write and test. When compared with the four days it took to write the applet's initial buggy prototype, it may not seem to have been worth the time or effort. In addition, it takes more skilled programmers to develop truly reusable components, and taking this approach meant that we could no longer rely solely on undergraduates, our previous source of programmers.

If a digital library of reusable learning objects is ever to become a reality, however, we must continue to invest the upfront time. In our case, although we took four months to design, implement, and test a single applet, we also produced several dozen components that we consider to be stable and generally reusable. Indeed, they were easily reused by the undergraduate who programmed the radiosity applet. For us, the long-term benefits of this far outweigh the short-term gains of the hacked-together prototype, whose code was not reusable.

We believe the complete decomposition approach we have adopted for component creation allows us to offer full customization of components by content authors, thereby increasing the degree to which individual components are reused. We are also now convinced that the effort and time we must devote to designing and developing not only good components but also a large amount of underlying infrastructure will enable us to reap huge rewards down the line and significantly enhance the field of educational software component technology.

Although starting a good repository of reusable learning objects is a time-consuming and expensive task, ultimately doing so should dramatically reduce the time needed to create interactive learning environments. Not only will mature programmers' time be reduced but inexperienced developers will be able to make sophisticated learning environment by re-using learning objects that encapsulate advanced functionality that they would not be able to easily program on their own.

Developers' new creations and the new components that are a part of them, can, in turn, be contributed to the library, creating a snowball effect of additional content, both in complete instructional applications and in new building blocks from which future applications can be built.

5. ACKNOWLEDGMENTS

We would like to acknowledge helpful comments of Andy van Dam and contributions to the Exploratories project from Rosemary Michelle Simpson. We would also like to thank our sponsors, the National Science Foundation, through our NSDL and STC grants, and Sun Microsystems.

6. REFERENCES

[1] IBM. alphaBeans: JavaBeans by IBM, http://alphaworks.ibm.com/alphabeans/

[2] Jeff E. Beall, Adam M. Doppelt, and John F. Hughes. "Developing an Interactive Illustration: Using Java and the Web to Make It Worthwhile," in Proceedings of 3D and Multimedia on the Internet, WWW and Networks, 16-18 April 1996, Pictureville, National Museum of Photography, Film & Television, Bradford, UK, 1996.

[3] BeanHaus. Java Bean Repository, http://www.beanhaus.org

[4] Stephanie Doublait. "Standard Reuse Practices: Many Myths vs. a Reality," in Standard View, Vol.5, No. 2, June, 1997.

[5] EOE Foundation. Educational Objects Economy: Building Communities that Build Knowledge, http://www.eoe.org.

[6] ESCOT project. Educational Software Components of Tomorrow, http://www.sri.com/policy/ctl/html/escot.html.

[7] E-Slate project. An exploratory learning environment, http://e-slate.cti.gr/

[8] Exploratories project. Web-based educational software, http://www.cs.brown.edu/exploratory/

[9] Exploratorium. The San Francisco Exploratorium: museum of science, art, and human perception, http://www.exploratorium.edu/

[10] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes. "Computer Graphics: Principles and Practice," Addison-Wesley, 1996, ISBN 0-201-84840-6.

[11] Richard P. Gabriel. "Patterns of Software: Tales from the Software Community," Oxford University Press, August 1996.

[12] R. F. Gurwitz and R. W. Thorne and A. van Dam and I. B. Carlbo. "BUMPS: A Program for Animating Projections," in Proceedings of ACM SIGGRAPH '80, pp. 231-237, 1980.

[13] IEEE. "Web-Based Learning and Collaboration" special issue, Computer, Vol. 32, No. 9, September 1999.

[14] IMS Global Learning Consortium, Inc., http://www.imsproject.org/

[15] IrYdium Project. Java Enhanced Chemical Education, http://ir.chem.cmu.edu/irProject/

[16] JavaBeans. "Specification for the Java 2 Platform," http://java.sun.com/products/javabeans/glasgow/

[17] Rosemary Michelle Simpson, Anne Morgan Spalter, and Andries van Dam. "Exploratories: An Educational Strategy for the 21st Century," in ACM SIGGRAPH '99 Conference Abstractions and Applications, 1999.

[18] Anne Morgan Spalter, Michael LeGrand, Saori Taichi, and Rosemary Michelle Simpson. "Considering a Full Range of Teaching Techniques for Use in Interactive Educational Software: A" Practical Guide and Brainstorming Session, in Proceedings of IEEE FIE 2000 (Frontiers in Education), October 2000.

[19] Anne Morgan Spalter and Rosemary Michelle Simpson. "Reusable Hypertext, Structures for Distance and JIT Learning," in Proceedings of ACM Hypertext 2000, June 2000.

[20] Anne Morgan Spalter and Rosemary Michelle Simpson. "Integrating Interactive Computer-Based Learning Experiences Into Established Curricula," in Proceedings of ACM ITICSE 2000 (Innovation and Technology in Computer Science Education), July 2000.